SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

ONEDU-PALVELUN KÄYTTÄJIEN JA KÄYTTÖOIKEUKSIEN HALLINTA

Mobie Oy

TEKIJÄ/T: Joni Eskelinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Joni Eskelinen	
Työn nimi OnEdu-palvelun käyttäjien ja käyttöoikeuksien hallinta	
Päiväys 27.4.2013	Sivumäärä/Liitteet 40/0
Ohjaaja(t) lehtori Keijo Kuosmanen	
Toimeksiantaja/Yhteistyökumppani(t) Mobie Oy	

Tiivistelmä

Tässä opinnäytetyössä suunniteltiin ratkaisu OnEdu-palvelun käyttäjien ja käyttöoikeuksien hallintaan sekä toteutettiin hallintasovelluksen sovelluskehys.

OnEdu on Mobie Oy:n kehittämä, oppilaitoksille suunnattu verkkopalvelu, jota käytetään monikanavaisten oppimateriaalien tuottamiseen ja jakeluun sekä viestintään. Palvelukokonaisuus koostuu useista itsenäisistä verkkopalveluista, jotka voivat olla hajautettu eri verkko-osoitteiden alle sekä fyysisesti eri palvelimille.

Organisaatiokohtaiseen OnEduun kuuluvien hajautettujen palveluiden käyttäjiä ja käyttöoikeuksia haluttiin hallinnoida keskitetysti. Tätä varten tarvittiin räätälöity sovellus, jonka tekemisestä saatiin rajattua osa tätä opinnäytetyötä varten.

Opinnäytetyössä perehdyttiin ongelmaan ja suunniteltiin käyttöoikeusmalli. Lisäksi toteutettiin ohjelmistokehys sovelluksen pohjaksi. Sovellus on verkkopohjainen ja sen ohjelmointiin käytettiin PHP-kieltä. Tietokanta suunniteltiin käytettäväksi MySQL-ohjelmistossa.

Lopputuloksena löydettiin ratkaisu sopivaksi tietomalliksi käyttäjien ja käyttöoikeuksien hallintaan, suunniteltiin alustava tietokantaskeema sekä toteutettiin ohjelmistorunko, jonka päälle sovelluksen rakentaminen jatkuu opinnäytetyön valmistumisen jälkeen.

Avainsanat
verkkosovellus, käyttäjätili, käyttöoikeus, tietoturva, tietokanta, PHP

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Joni Eskelinen			
Title of Thesis Management of Users and Permissions in OnEdu-Service			
Date	27 April 2013	Pages/Appendices	40/0
Supervisor(s) Mr Keijo Kuosmanen, lecturer			
Client Organisation /Partners Mobie Oy			

Abstract

The objective of this thesis was to design a solution for the purpose of managing users and permissions in the OnEdu-service, as well as to implement a framework for the management application.

OnEdu is an online service developed by Mobie Oy. It is aimed for educational institutions and is used for publication and distribution of digital multiplatform educational materials as well as for communication. The whole service package consists of several independent online services which may be scattered over multiple domains and on different servers.

The management of users and permissions in per organization OnEdu services was preferred to be centralized. Tailor-made software was required for the purpose, and a fraction of it was delimited for the topic of this thesis.

In this thesis the problem was explored and a permission model was designed. In addition, an application framework was implemented to serve as a basis for the final software. The application is web-based and it was programmed using the PHP language. The database was designed to be used with MySQL server software.

As a result a solution for a suitable model for the management of users and permissions was found, rudimentary database scheme was designed and an application framework was implemented. The final software will be constructed later on top of the framework after the thesis project is finished.

Keywords
web application, user account, access right, information security, database, PHP

ESIPUHE

Tämä opinnäytetyö on tehty Savonia-ammattikorkeakoulun tietotekniikan koulutusohjelmassa. Haluan kiittää Mobie Oy:n toimitusjohtaja Kari Vatasta puitteista, jotka mahdollistivat päättötyön ja tutkinnon suorittamisen työelämän ohessa.

Kuopiossa 27.4.2013

Joni Eskelinen

SISÄLTÖ

TERMIT JA LYHENTEET

ACL	Access Control List, pääsylista. Suojauksen kohteeseen liitetty lista käyttöoikeuksista.
AD	Active Directory. Windows-toimialueen käyttäjätietokanta ja hakemistopalvelu.
API	Application Programming Interface, ohjelmointirajapinta. Määrittely jonka kautta ohjelmistot voivat kommunikoida keskenään.
CSS	Cascading Style Sheets. Kieli jolla kuvataan tyylit jollain muulla kuvauskielellä kirjoitetuille dokumenteille, kuten HTML-verkkosivut.
FQN	Fully Qualified Name. Yksiselitteinen nimi objektille, kuten ohjelmistoluokalle tai tiedostopolulle.
HTML	HyperText Markup Language. Verkkosivujen kuvauskieli.
HTTP(S)	Hypertext Transfer Protocol (Secure). Hypermedian siirtoon käytetty verkkoprotokolla. HTTPS siirtää tiedon salattuna.
IDE	Integrated Development Environment, ohjelmointiympäristö.
JavaScript	ECMAScript-standardiin perustuva ohjelmointikieli, jota käytetään mm. verkkosivujen dynaamisen toiminnallisuuden toteuttamiseen.
LDAP(S)	Lightweight Directory Access Protocol (sekä Secure LDAP). Hakemistopalvelujen, kuten AD, käyttöön tarkoitettu verkkoprotokolla.
MVC	Model-View-Controller, malli-näkymä-käsittelijä. Ohjelmistoarkkitehtuurityyli, jossa käyttöliittymä erotetaan sovelluslogiikasta.
PHP	PHP: Hypertext Preprocessor. Ohjelmointikieli, jota käytetään usein verkkosivujen luonnissa.
SOAP	Simple Object Access Protocol. Verkkoprotokolla, jota käytetään sovellusten väliseen tiedonvälitykseen ja etäproseduurien kutsumiseen.
SQL	Structured Query Language. Kyselykieli, jolla relaatiotietokantaan voidaan tehdä hakuja, muutoksia ja lisäyksiä.

SSO	Single Sign-On, kertakirjautuminen. Menetelmä, jolla pääsy useisiin järjestelmiin voidaan sallia yhdellä kirjautumisella.
URI	Uniform Resource Identifier. Merkkijono, jolla yksilöidään jokin tieto.
URL	Uniform Resource Locator. Merkkijono, jolla ilmaistaan jonkin tiedon sijainti. Verkko-osoitteet ovat URL:eja.
WYSIWYG	What You See Is What You Get. Graafinen sisältöeditori, jossa muokattava sisältö näytetään siinä muodossa kuin se esitetään.

1 JOHDANTO

Opinnäytetyön tavoitteena on löytää ratkaisu OnEdu-palvelun käyttäjien ja käyttöoikeuksien hallintaan. OnEdu on Mobie Oy:n kehittämä oppilaitoksille suunnattu verkkopalvelu, jota käytetään monikanavaisten oppimateriaalien tuottamiseen ja jakeluun sekä viestintään. Mobie Oy on kuopiolainen ohjelmistoalan yritys, joka on perustettu vuonna 2010.

OnEdu on tuote, johon kuuluu useita verkkopalveluja hajautettuna. Kuitenkin kaikkien hajautettujen järjestelmien käyttäjät ja käyttöoikeudet tulisi pystyä hallinnoimaan keskitetysti. Käyttökohteita ovat mm. oppimateriaalien käyttö ja tuotanto, verkkosivujen ylläpito sekä järjestelmien hallinnointi. Käyttäjiä ovat mm. oppimateriaalin tuottajat, opettajat, opiskelijat, verkkosivuston ylläpitäjät sekä järjestelmän hallinnoijat. Eri käyttökohteilla voi olla eri tarpeita käyttöoikeuksista, joten käyttöoikeudet täytyy saada rajattua palvelu- ja käyttäjäkohtaisesti.

Tämän opinnäytetyön tekemisen aikana OnEdu on vielä kehitysohjelma, jossa on mukana useita oppilaitoksia ja yhteistyökumppaneita. Kehitysohjelman tavoitteena on löytää parhaat käytännöt ja kartoittaa pedagoginen soveltuvuus. Tämän vuoksi myös käyttäjien ja käyttöoikeuksien hallinnan tarpeet tulevat luultavasti elämään, joten suunnittelussa on huomioitava järjestelmän joustavuus ja skaalautuvuus.

Koska järjestelmää ei kokonaisuudessaan ollut mahdollista saada opinnäytetyölle varatussa ajassa valmiiksi, on aihe rajattu siten, että tässä raportissa keskitytään järjestelmän suunnitteluun sekä ohjelmistokehyksen toteutukseen. Järjestelmän kehitys jatkuu vielä opinnäytetyön valmistuttua.

Raportin alussa esitellään ongelma, johon opinnäytetyöllä haetaan ratkaisua. Tämän jälkeen esitellään käyttöoikeusmalli, jota käyttöoikeuksien hallinnassa sovelletaan. Lopuksi perehdytään järjestelmää varten luodun ohjelmistokehyksen ydinkomponentteihin ja niiden toimintaan.

2 TYÖN LÄHTÖKOHDAT

Mobie Oy kehittää oppilaitoksille suunnattua järjestelmää digitaalisten, useilla päätelaitteilla toimivien oppimateriaalien tuottamiseen ja levitykseen. OnEdu-palvelukonseptia on ideoitu vuodesta 2012 lähtien muutamien oppilaitosten ja yhteistyökumppanien avustuksella. Vuoden 2013 alussa käynnistettiin OnEdu-kehitysohjelma, johon valittiin useita kouluja eri puolelta Suomea. Kehitysohjelman tavoite on kehittää koulujen oppimis- ja viestintäratkaisu, joka hyödyntää Mobie Oy:n olemassa olevia tuotteita. Lisäksi kehitysohjelman aikana pyritään löytämään järjestelmän parhaat käytännöt ja kartoittamaan pedagoginen soveltuvuus sekä kehittämään järjestelmää näistä lähtökohdista.

OnEdu nojaa vahvasti ilmaiseen avoimen lähdekoodin WordPress-julkaisujärjestelmään[1] ja siihen tehtyihin lisäosiin. Osa käytettävistä lisäosista on niin ikään kolmannen osapuolen avointa lähdekoodia, kun taas osa on Mobien tarkoitusta varten räätälöimiä.

Oppimateriaalit tehdään Mobie Zine -tuotteella. Mobie Zine on WordPress-lisäosa, joka mahdollistaa responsiivisten sähköisten verkkojulkaisujen tuottamisen. Responsiivisuus tarkoittaa, että sama sisältö mukautuu käytettävän päätelaitteen mukaan, ts. sisältö skaalautuu näytölle. Zine-julkaisut ovat luettavissa tietokoneilla, tableteilla, älypuhelimilla sekä televisioilla, joissa on Internet-selain. Mobie Zine perustuu HTML5-standardiin, joten se toimii millä tahansa päätelaitteella, jossa on HTML5:tä ymmärtävä selainohjelmisto. Mobie Zineä on kehitetty vuodesta 2011 lähtien.

Oppimateriaaleihin voidaan lisätä erilaisia rikasteita, kuten videoita ja upotuksia ulkoisista palveluista. Kolmannen osapuolen tuottamat upotettavat palvelut ovat lähtökohtaisesti julkisia, vaikka pääsy oppimateriaalijulkaisuun itseensä olisikin rajattu tunnistautumisella. Esimerkiksi YouTube-videoiden upotus suojattuun julkaisuun ei tee videoista suojattuja. Joidenkin rikasteiden näkyvyyden rajaus tunnistautumisella voi tulla tulevaisuudessa tarpeelliseksi tapauksissa, joissa se on teknisesti mahdollista toteuttaa, joten asia on otettava kehityksessä huomioon.

Palvelukokonaisuuden hahmotuttua ja ominaisuuksien lisäännyttyä havaittiin, että käyttäjien hallinta voi koitua ongelmalliseksi. Palveluun kuuluu useita verkkopohjaisia järjestelmiä, jotka voivat olla hajautettu eri verkko-osoitteisiin sekä fyysisesti eri palvelimille. Lisäksi eri järjestelmissä on omat itsenäiset käyttäjähallintansa, joten käyttäjiä täytyisi ylläpitää kaikkialla erikseen. Koettiin siis tarpeelliseksi kehittää järjestelmä, jolla käyttäjiä ja

[1] Tietoa WordPressistä: http://wordpress.org/about/.

käyttöoikeuksia pystytään hallitsemaan keskitetysti ja tietoturvallisesti kaikkiin OnEduun kuuluviin järjestelmiin.

Koska eri järjestelmillä on eri käyttötarkoitukset ja siten eri vaatimukset käyttöoikeuksien suhteen, oli alusta asti selvää, että käyttäjien ja käyttöoikeuksien hallinnan täytyi olla joustava ja laajennettavissa eri käyttökohteisiin. Käyttöoikeuksien tietomallin oli oltava sellainen, että sillä pystytään kuvaamaan ja ylläpitämään käyttäjien käyttöoikeuksia erilaisiin järjestelmiin. Lisäksi käyttäjien ja käyttöoikeuksien hallinta täytyi olla helposti integroitavissa muihin järjestelmiin.

Toivottiin myös, että käyttäjät voitaisiin synkronoida ulkoisesta tietovarastosta, sillä usein oppilaitoksilla on jo käytössä järjestelmä, jossa käyttäjiä ylläpidetään. Tyypillisesti tällainen tietovarasto on Active Directory. Tällaisessa tapauksessa käyttäjät todennetaan ulkoisen tietovaraston käyttäjätietoja vastaan, mutta käyttöoikeuksia ylläpidettäisiin kehitettävässä järjestelmässä.

Käyttömukavuuden kannalta oli myös toivottavaa, että järjestelmä tukisi kertakirjautumista eli että käyttäjän ei tarvitse syöttää käyttäjätunnusta ja salasanaa joka palveluun erikseen, vaan yhteen järjestelmään todentautuminen kirjaa käyttäjän samalla kaikkiin käyttöoikeuksien sallimiin palveluihin.

3 ONGELMAN KUVAUS

OnEdu on organisaatiokohtainen hajautettu järjestelmä, jota useat henkilöt käyttävät eri käyttötarkoituksiin. Eri käyttötarkoitukset edellyttävät, että käyttäjille voidaan määritellä eri käyttöoikeuksia. Käyttäjien ja käyttöoikeuksien hallintaa varten tarvitaan järjestelmä, josta käyttäjätiedot ovat keskitetysti saatavissa, jonka avulla käyttäjät voidaan todentaa ja josta saadaan valtuutustiedot eri toimintoihin muissa järjestelmissä. Hajautus edellyttää, että hallinta toimii itsenäisenä järjestelmänä, jonka rajapintojen kanssa muut järjestelmät kommunikoivat. Käyttäjien ja käyttöoikeuksien soveltamiseen tarvitaan myös integraatio kaikkiin niihin järjestelmiin, jotka halutaan toiminnallisuuden piiriin.

3.1 Käyttötarkoitukset

OnEdu sisältää asiakaskohtaisesti eri palveluita kunkin asiakkaan tarpeiden ja organisaation laajuuden mukaan. Käyttökohteita ovat mm. koulujen verkkosivujen ylläpito WordPress-julkaisujärjestelmällä, oppimateriaalien tuotanto Mobie Zine -järjestelmällä, oppimateriaalien lukeminen ja oppimateriaaleihin liitettyjen rikasteiden käyttö. Koska käyttötarkoituksia voi tulla jatkossa lisää, täytyy järjestelmä suunnitella siten, että uusien palveluiden integrointi on mahdollista ja mahdollisimman yhdenmukaista.

Eri palveluilla on erilaiset vaatimukset käyttöoikeuksien suhteen. Järjestelmään tulee voida luoda palvelukohtaisia käyttöoikeuksia, joita sovelletaan myönnettäessä valtuuksia käyttäjille kyseiseen palveluun. Esimerkiksi käyttöoikeus *”saa luoda WordPress-sivuja”* on hyödyllinen ainoastaan WordPress-kontekstissa eli silloin, kun käyttöoikeuden kohteena on WordPress-julkaisujärjestelmä. Kyseistä käyttöoikeutta ei tulisi siis olla myönnettävissä, kun kohteena on jokin muu resurssi, esimerkiksi oppimateriaalien lukeminen.

Organisaation (tässä tapauksessa koulun) koon mukaan kaikkia palveluja ei välttämättä oteta käyttöön tai niiden käyttöaste voi vaihdella. Lisäksi suurempi oppilaitos voi tarvita useamman verkkosivuston, jotka voi olla mielekkäämpää erottaa omiksi itsenäisiksi julkaisujärjestelmikseen omien verkko-osoitteiden alle. Kuitenkin kaikkien saman organisaation julkaisujärjestelmien käyttöoikeuksia tulisi pystyä hallitsemaan keskitetysti. Vaikka WordPressissä on joustava roolipohjainen käyttöoikeuksien hallinta, ei kuitenkaan voida puhua ns. enterprise-tason julkaisujärjestelmästä, jossa työnkulku ja kunkin työvaiheen käyttöoikeudet olisivat yksityiskohtaisesti rajattavissa tietyille käyttäjille. Tästä syystä onkin perusteltua hajauttaa erilliset sivustot omiin WordPress-instansseihinsa, jolloin voidaan rajata pääsy käyttäjäkohtaisesti vain tietyn julkaisujärjestelmän hallintaan.

OnEdu-palvelukokonaisuus on siis hajautettu jo yksittäisen organisaation käytössä. Hajautus voi tarkoittaa useampia WordPress-instansseja, jotka saattavat sijaita fyysisesti eri

palvelimilla. Lisäksi eri sivustot toimivat eri verkko-osoitteiden (domain) alla, joten kirjautumistiedon tallennus asiakaspäässä (loppukäyttäjän Internet-selain) aiheuttaa omat haasteensa, sillä evästeiden asetus ei ole mahdollista kuin senhetkisen verkko-osoitteen alaisuuteen. Hajautuksen vuoksi järjestelmän täytyy siis toimia itsenäisesti ja muiden siihen liittyvien järjestelmien täytyy kommunikoida käyttäjien ja käyttöoikeuksien hallinnan kanssa verkkopohjaisten rajapintojen – kuten www-sovelluspalvelun (engl. web service) – kautta.

3.1.1 Verkkosivujen ylläpito

Koulun julkisten verkkosivujen ylläpito OnEdu:ssa tapahtuu WordPress-julkaisujärjestelmällä. Ylläpidosta voi vastata useampi kuin yksi henkilö, ja näillä voi olla eri käyttöoikeuksia. Jollekin henkilölle voidaan haluta vaikkapa antaa oikeudet luoda uutisia ja tiedotteita, mutta sama henkilö ei saa muokata sisältösivuja eikä muiden luomia uutisia eikä tiedotteita.

WordPressissä on lähtökohtaisesti oma käyttäjähallinta ja roolipohjainen käyttöoikeuksien hallinta. Integraatio ulkoiseen käyttäjähallintaan on mahdollista tehdä lisäosana. WordPressissä on kattavat valmiudet lisäosien ohjelmointiin, ilman että itse julkaisujärjestelmän lähdekoodiin tarvitsee tehdä muutoksia.

Koko WordPressin käyttäjäskeema olisi mahdollista korvata integraatiolla ulkoiseen käyttäjä- ja käyttöoikeushallintaan, ja siten välttää käyttäjätietojen säilyttäminen julkaisujärjestelmän alla. Kuitenkin useat avoimesti saatavilla olevat lisäosat eivät ymmärrä tätä, vaan käyttävät suoraan WordPressin käyttäjätietokantaa SQL-kyselyillä, sen sijaan että käyttäjien tietoja haettaisiin WordPressin API:n kautta. Tästä syystä yhteensopivuuden varmistamiseksi toteutus kannattaa tehdä siten, että käyttäjän kirjautuessa WordPressiin tiedot ja käyttöoikeudet synkronoidaan OnEdun käyttäjähallinnasta. Tällöin kyseistä WordPress-instanssia käyttäneiden käyttäjien tunnukset ja käyttöoikeudet löytyvät myös ko. WordPressin tietokannasta, joten yhteensopivuusongelmia lisäosien kanssa ei pitäisi ilmetä.

Käyttäjien ja käyttöoikeuksien synkronoinnissa OnEdusta WordPressiin on otettava huomioon skeemojen yhteensopivuus. Esimerkiksi käyttöoikeuksien synkronoinnissa voi olla tarpeen käyttää ns. assosiaatiotaulua eli tietorakennetta, jolla saadaan sidottua tietty arvo toiseen arvoon. Lisäksi salasanojen synkronointi ei ole mahdollista, mikäli salasanat halutaan tallentaa OnEdussa tiivisteinä (engl. hash). Tämä tosin ei pitäisi osoittautua ongelmaksi, sillä kokemuksestani lisäosat pääasiassa käyttävät WordPressin API:a käyttäjien todennukseen, jolloin todennuslogiikka voidaan integraatiolisäosalla ylikirjoittaa käyttämään keskitettyä käyttäjienhallinnan rajapintaa.

3.1.2 Oppimateriaalien tuotanto

Oppimateriaalit tuotetaan Mobie Zine -järjestelmällä. Mobie Zine on WordPress-lisäosa, joka hyödyntää WordPressin sisällönhallintaominaisuuksia, kuten graafista WYSIWYG-editoria ja mediakirjastoa kuvien hallintaan.

Koska Mobie Zine toimii WordPressin alla, sovelletaan siihen WordPressin roolipohjaista käyttöoikeuksien hallintaa. Zine-julkaisujen luomiseen valtuuttavat käyttöoikeudet kuuluvat siis samaan kontekstiin kuin verkkosivujen ylläpito, vaikkakin voidaan puhua kahdesta erillisestä käyttökohteesta. Todennäköisesti oppimateriaaleille käytännössä luodaankin oma WordPress-instanssi, johon oppimateriaalien tuottajille annetaan käyttöoikeudet. Näin saadaan pidettyä verkkosivut ja oppimateriaalit täysin eriytettyinä, mikä on tietoturvan kannalta parempi vaihtoehto.

Oppimateriaaleihin voidaan liittää erilaisia rikasteita, kuten videoita sekä upotuksia muista verkkopalveluista. Esimerkiksi oppituntitallenteet luodaan erillisellä järjestelmällä ulkoiselle palvelimelle. Mikäli tuntitallenteita halutaan liittää julkaisuun, täytyy myös niiden käyttöoikeudet ottaa huomioon.

Kaikissa tapauksissa käyttöoikeuksia ei voida soveltaa upotettavaan materiaaliin, vaan upotuksien täytyy olla julkisesti saatavilla. Esimerkiksi YouTube-videoita tai muuta kolmannen osapuolen palveluissa olevaa materiaalia ei voi rajata käyttöoikeuksin.

3.1.3 Oppimateriaalien lukeminen

Oppimateriaalit tuotetaan pääasiassa oppilaiden käyttöön. Oppimateriaalit voivat olla julkisia tai kaupallisia. Jälkimmäisessä tapauksessa materiaalien näkyvyys täytyy pystyä rajaamaan siten, että vain henkilöt, joilla on kuhunkin oppimateriaaliin lukuoikeus, voivat niitä tarkastella.

Lukuoikeus voi esimerkiksi tarkoittaa sitä, että henkilö on ostanut lisenssin sähköiseen julkaisuun tai että koulu on myöntänyt yhden opetusryhmän oppilaille oikeuden käyttää tiettyä julkaisua. Käyttöoikeus yksittäiseen opetusmateriaaliin olisi mielekästä pystyä myöntämään ryhmittäin, ei vain yksittäisille henkilöille. Tällöin lukuoikeus voitaisiin myöntää kerralla kokonaiselle opetusryhmälle tai luokalle. Käyttäjähallintaan täytyy siis pystyä luomaan ryhmiä, joihin voidaan liittää useita käyttäjiä. Käyttöoikeudet voidaan tällöin asettaa ryhmälle, jolloin ne periytyvät kaikille ryhmään liitetyille käyttäjille.

3.1.4 Käyttäjien ja käyttöoikeuksien ylläpito

Käyttäjien ja käyttöoikeuksien hallinta täytyy myös pystyä rajaamaan käyttäjittäin. Hallintaan valtuuttavien käyttöoikeuksien ylläpito olisi johdonmukaista myös pitää samassa jär-

jestelmässä. Tällöin hallintaan valtuutettujen käyttäjien käyttöoikeuksia täytyy myös pystyä rajaamaan yksityiskohtaisemmin. Esimerkiksi käyttäjät eivät saa pystyä myöntämään itselleen enempää käyttöoikeuksia eivätkä luomaan uusia käyttäjiä kattavammilla käyttöoikeuksilla kuin heillä itsellään on. Tyypillinen käyttötarkoitus voisi olla, että opettaja lisää kurssiryhmälleen oikeudet oppimateriaalijulkaisuun, jonka hän on itse tehnyt. Opettaja ei saisi kuitenkaan pystyä liittämän käyttöoikeuksia oppimateriaaleihin, jotka eivät ole hänen tekemiään ts. joihin hänellä ei lähtökohtaisesti ole käyttöoikeutta.

3.2 Tekniset vaatimukset

Järjestelmän tulee tarjota käyttäjien hallinta, käyttöoikeuksien hallinta sekä käyttäjien todennus. Lisäksi järjestelmässä tapahtuvista toimista olisi hyvä kirjata lokimerkinnät, jotta virhetilanteet ja mahdolliset väärinkäytökset ovat tarvittaessa selvitettävissä.

Useimmissa tapauksissa oppilaitoksilla on jo erillinen käyttäjätietokanta, esim. Active Directory (AD). Tällöin ei ole tarkoituksenmukaista ylläpitää käyttäjätietoja salasanoineen kahdessa erillisessä järjestelmässä. Järjestelmä täytyy siis olla integroitavissa ulkoiseen tietovarastoon.

Koska OnEdu-palvelukokonaisuus on hajautettu, olisi mielekästä, ettei käyttäjän tarvitse kirjautua jokaiseen palveluun erikseen vaan tarjolla olisi kertakirjautuminen (engl. single sign-on, SSO). Kertakirjautumisella tarkoitetaan sitä, että kun käyttäjä tunnistautuu kerran onnistuneesti, hän voi sen jälkeen käyttää kaikkia palveluja käyttöoikeuksien rajoittamalla tavalla, ilman että käyttäjätunnusta ja salasanaa tarvitsee syöttää uudelleen.

3.2.1 Käyttäjien hallinta

Käyttäjien hallinnalla tarkoitetaan, että järjestelmässä ylläpidetään käyttäjätietoja. Käyttäjätietoja ovat käyttäjätunnus ja salasana sekä mahdolliset metatiedot, kuten etu- ja sukunimi. Kaikkia tietoja ei välttämättä säilytetä järjestelmässä, mikäli käytetään ulkoista tietovarastoa, kuten AD:ta. Jokaisesta käyttäjästä on kuitenkin tallennettava vähintään käyttäjätunnus, jotta käyttäjään voidaan liittää käyttöoikeuksia.

Käyttäjät yksilöidään järjestelmässä uniikilla tunnisteella. Tunniste on tekstimuotoinen käyttäjätunnus, jota yhdessä salasanan kanssa käytetään henkilön todentamiseen. Käyttäjätunnuksen muodolle ei voi asettaa rajoituksia, mutta täytyy ottaa huomioon, että eri järjestelmät voivat edellyttää tietynmuotoisia käyttäjätunnuksia. Jokin palvelu saattaa vaatia, että käyttäjätunnus on sähköpostiosoite, mutta toinen ei salli tunnuksessa erikoismerkkejä. Tällöin samaa tunnusta ei voida käyttää molemmissa palveluissa. Käyttäjienhallinnassa täytyy siis olla mahdollisuus liittää samaan käyttäjään useampi käyttäjätunnus, jotta tunnistautumisen integrointi eri järjestelmiin on mahdollista.

Käyttöoikeudet liitetään ainoastaan käyttäjän ensisijaiseen käyttäjätunnukseen. Mikäli käyttäjä kirjautuu toissijaisella tunnuksellaan, täytyy kirjautumislogiikan selvittää käyttäjän ensisijainen käyttäjätunnus ja huomioida tähän liitetyt käyttöoikeudet.

Käyttäjät voivat kuulua useampiin käyttäjäryhmiin tai olla kuulumatta yhteenkään. Käyttöoikeuksia tulee voida myöntää joko yksittäiselle käyttäjälle tai käyttäjäryhmälle, jolloin kaikki ryhmään kuuluvat käyttäjät perivät kyseisen käyttöoikeuden. Lisäksi voi tulla tarve poikkeuksille. Esimerkiksi ryhmän yhdeltä käyttäjältä voidaan haluta poistaa tietty käyttöoikeus, joka muutoin annetaan koko ryhmälle. Käyttöoikeuksia täytyy siis pystyä liittämään käyttäjiin sallivina tai epäävinä, joista jälkimmäinen poistaa kyseisen käyttöoikeuden mikäli se toista kautta käyttäjälle myönnettäisiinkin.

Mikäli ulkoista tietovarastoa käyttäjien todennukseen ei käytetä, täytyy järjestelmään tallentaa myös käyttäjän salasana. Salasanat tallennetaan tietoturvallisesti yksisuuntaisena suolattuna tiivisteenä (engl. salted hash). Salasanaformaatti olisi kuitenkin hyvä merkitä käyttäjätietoihin, jotta jatkossa on mahdollista käyttää eri formaatteja sekaisin, mikäli käyttäjätietoja tuodaan muualta eräajona salasanojen kera.

3.2.2 Käyttöoikeuksien hallinta

OnEdu koostuu erilaisista resursseista, joihin pääsy täytyy rajata käyttöoikeuksin. Resurssilla tarkoitetaan siis mitä tahansa asiaa, johon täytyy soveltaa käyttäjien valtuutusta. Resursseja on erityyppisiä ja näillä on eri vaatimukset käyttöoikeuksien suhteen. Lisäksi yhden resurssityypin käyttöoikeudet ovat merkityksettömiä toiselle resurssityypille.

Resurssityyppejä voi olla esimerkiksi WordPress-julkaisujärjestelmä, oppimateriaalijulkaisu, verkkokauppa sekä myös käyttäjien ja käyttöoikeuksien hallintajärjestelmä itsessään. Resurssiksi kutsutaan yksittäistä ilmentymää tietystä resurssityypistä.

Käyttöoikeudet ovat resurssityyppikohtaisia. Tietyn resurssityypin resurssille ei tule voida myöntää käyttöoikeuksia, jotka kuuluvat toiselle resurssityypille. Järjestelmään tulee voida määritellä käyttöoikeuksia erikseen jokaiselle resurssityypille. Kun valtuutuksen piiriin lisätään uusi resurssi, sen täytyy olla tiettyä resurssityyppiä ja näin ollen siihen voidaan myöntää vain kyseisen resurssityypin mukaisia käyttöoikeuksia.

Käyttäjälle resurssiin myönnettävät käyttöoikeudet voivat olla sallivia tai epääviä. Tällöin voidaan luoda poikkeuksia tapauksissa kun joukko käyttäjiä perii käyttöoikeuden ryhmäassosiaation kautta, mutta yksittäiseltä henkilöltä halutaan kyseinen käyttöoikeus poistaa.

Uusien resurssityyppien ja niihin liittyvien käyttöoikeuksien lisäämisen tulisi olla helppoa, jotta uusien palveluiden integrointi tunnistuksen ja valtuutuksen piiriin onnistuu tulevai-

suudessa. Samoin uusien resurssien luominen täytyy olla yksinkertaista, ja luultavasti tarpeellista myös rajapinnan kautta. Esimerkiksi kun luodaan uusi oppimateriaalijulkaisu, tieto tästä olisi hyvä saada automaattisesti myös käyttöoikeuksien hallintaan, jotta resurssiin voidaan liittää käyttöoikeuksia halutuille käyttäjille.

3.2.3 Integraatio ulkoiseen tietovarastoon

Usein oppilaitoksien käyttäjiä ja kirjautumistunnuksia hallitaan jo erillisessä järjestelmässä, mutta halutaan että samoilla tunnuksilla voi myös käyttää kaikkia OnEduun liittyviä palveluja käyttöoikeuksien rajoittamalla tavalla. Järjestelmä täytyy siis olla integroitavissa ulkoiseen tietovarastoon.

Integraatiot tulevat lähtökohtaisesti olemaan yksisuuntaisia. Tällä tarkoitetaan sitä, että synkronointia ei tapahdu OnEdun käyttäjähallinnasta ulospäin, vaan tiedot tuodaan ulkoisesta tietolähteestä OnEduun. Ulkoinen tietovarasto toimii siis ns. master datana eli perustietona.

Synkronoinnin ei normaalitapauksessa tarvitse tapahtua reaaliaikaisesti, vaan tiedot voidaan päivittää eräajona esimerkiksi kerran päivässä. Lisäksi yksittäisen käyttäjätunnuksen tiedot voidaan valinnaisesti synkronoida vasta sitten, kun tunnusta käytetään ensimmäistä kertaa, ts. kun tunnusta ei vielä löydy OnEdun käyttäjähallinnasta. Tällöin suoritetaan kirjautumisyritys ja onnistuttaessa haetaan tarvittavat tiedot rajapinnan kautta OnEdun tietokantaan.

Ulkoista tietovarastoa käytettäessä käyttäjien todennus tulisi myös tapahtua ulkoista järjestelmää vastaan. Tällöin salasanoja ei tarvitse synkronoida, eikä se olisi tietoturvankaan kannalta mielekästä. Lisäksi eri tietovarastoissa käytetään eri salasanaskeemoja eli salasanat ovat tallennettu eri muodossa, ja näistä jokainen eri menetelmä tulisi toteuttaa myös OnEdun käyttäjähallinnan todennuslogiikkaan. Kun käyttäjä todennetaan ulkoisesta tietovarastosta, ei tarvitse ottaa kantaa kyseisen järjestelmän salasanaskeemaan tai todennuslogiikkaan, vaan riittää että saadaan tieto kyseessä olevan käyttäjätunnuksen todennuksen onnistumisesta.

Koska todennus voi tapahtua ulkoista järjestelmää vastaan ja salasanaskeema on OnEdun näkökulmasta triviaali, täytyy salasanat kuljettaa selkokielisenä koko todennusprosessin läpi. Tämän vuoksi on suositeltavaa käyttää suojattua protokollaa rajapintojen väliseen tietoliikenteeseen. Rajapinnasta riippuen mahdollisia tekniikoita ovat mm. SOAP HTTPS-protokollalla sekä LDAPS, eli LDAP SSL-protokollalla.

3.2.4 Kertakirjautuminen

Koska OnEdu koostuu useista itsenäisistä järjestelmistä eri verkko-osoitteiden alla, täytyy käyttäjä todentaa näihin kaikkiin erikseen. Jotta vältyttäisiin käyttäjätunnuksen ja salasanan syöttämiseltä jokaiseen järjestelmään uudelleen, käyttäjien ja käyttöoikeuksien hallinnan täytyy tukea kertakirjautumista.

Kertakirjautuminen edellyttää, että kaikki kirjautumiset suoritetaan keskitetyn kirjautumispalvelun kautta. Kirjautumispalvelu säilyttää tiedon onnistuneesta tunnistautumisesta istunnossaan, joten palvelulta voidaan kysyä toiselle sivustolle siirryttäessä onko käyttäjä jo kirjautunut sisään. Mikäli käyttäjä on jo tunnistautunut, voi kirjautumista vaativa sivusto aloittaa oman istuntonsa käyttäjälle ilman käyttäjätunnuksen ja salasanan kysymistä.

Kertakirjautumisprosessi ei ota kantaa käyttöoikeuksiin, vaan huolehtii ainoastaan käyttäjän todentamisesta. Kertakirjautumista käyttävän palvelun täytyy erikseen huolehtia käyttöoikeuksien selvittämisestä käyttäjän todennuksen jälkeen.

3.2.5 Rajapinnat

Järjestelmän tulee tarjota rajapinnat, joiden avulla OnEdun palvelut voidaan integroida käyttäjien ja käyttöoikeuksien hallintaan. Tarvittavia operaatiota ovat käyttäjän todennus ja käyttäjän käyttöoikeuksien kysely haluttuun resurssiin. Lisäksi voi tulla tarpeelliseksi luoda resursseja käyttöoikeuksien hallintaan rajapinnan kautta sekä antaa käyttäjille käyttöoikeuksia haluttuun resurssiin.

Rajapinnat ovat luonteeltaan palvelin-palvelin-tyyppisiä (engl. server-to-server, s2s). Tämä tarkoittaa että järjestelmät keskustelevat keskenään rajapinnan kautta, eikä siten että käyttäjän päätelaite ottaa yhteyden palvelinrajapintaan, kuten asiakas-palvelin-mallissa (eng. client-server model). Poikkeuksena on kertakirjautumispalvelu, joka toimii asiakas-palvelin-periaatteella.

Palvelin-palvelin-rajapinnat toteutetaan verkkopohjaisina www-sovelluspalveluina. Koska rajapintojen kautta siirretään arkaluontoista tietoa, kuten salasanoja, täytyy verkkoliikenne olla asianmukaisesti salattu esimerkiksi TLS-protokollalla.

4 KÄYTTÖOIKEUSMALLI

Käyttöoikeusmallilla tarkoitetaan tietomallia, joka kuvaa käyttäjien, resurssien ja käyttöoikeuksien välisiä suhteita. Lisäksi käyttöoikeusmalli asettaa suuntaviivoja sille, kuinka tiedot olisi hyvä tallentaa pysyvään tietovarastoon. Ratkaisussa täytyi ottaa huomioon soveltuvuus eri käyttökohteisiin eli se että minkä tahansa OnEdu-resurssin käyttöoikeudet voidaan tallentaa ja ylläpitää järjestelmässä. Toisaalta taas ratkaisu olisi hyvä pitää mahdollisimman yksinkertaisena, ettei käyttäjien ja käyttöoikeuksien ylläpidosta tule tarpeettoman monimutkaista.

4.1 Tietoturvakonseptit

Käyttöoikeusmallia suunniteltaessa tutustuin eri tietoturvamalleihin ja käytäntöihin. Osin käytin myös omaa kokemustani aikaisemmista vastaavista järjestelmistä. Tässä luvussa on kuvattu lyhyesti muutamia konsepteja.

4.1.1 Pääsylista

Pääsylista (engl. access control list, ACL) on lista käyttöoikeuksista, jotka liitetään kohteeseen, kuten tiedostoresurssiin (What is ACL?). Monissa tiedostojärjestelmissä tiedostojen ja hakemistojen suojaus on toteutettu pääsylistoilla. Tiedostoon liitetty pääsylista voi antaa vaikkapa lukuoikeuden kaikille käyttäjille, mutta kirjoitusoikeuden vain yksittäisille.

4.1.2 Roolipohjainen pääsynrajaus

Roolipohjaisen pääsynrajauksen (engl. role-based access control, RBAC) ideana on, että käyttäjille myönnetään rooleja, joilla määritellään käyttöoikeudet. Roolit siis koostuvat käyttöoikeuksista. Roolit voivat lisäksi muodostaa roolihierarkian, jossa ylemmän tason rooli perii kaikki alemman tason roolien käyttöoikeudet. (An Introduction to Role-Rased Access Control.)

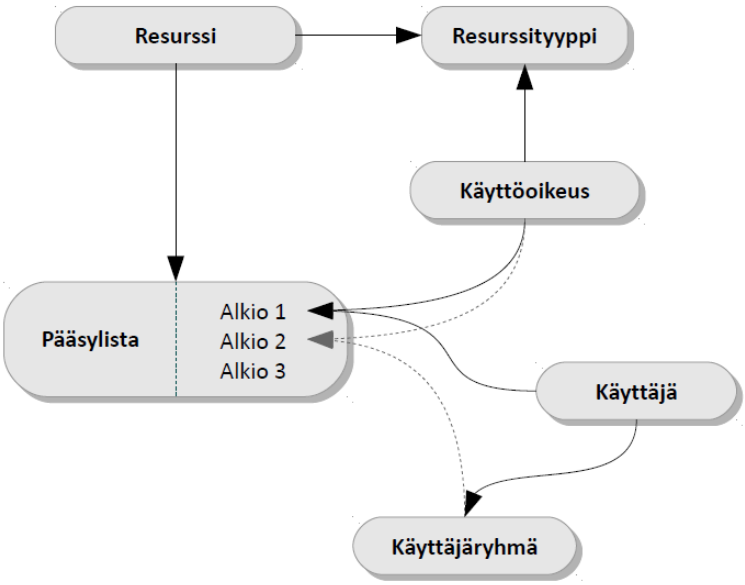
Koska käyttöoikeuksia ei aseteta suoraan käyttäjille vaan käyttäjät saavat käyttöoikeudet roolien kautta, tulee ylläpidosta yksinkertaisempaa, sillä yksittäisten käyttäjien käyttöoikeuksien ylläpitämisen sijaan muutoksia tarvitsee tehdä vain rooleihin.

4.2 Valittu ratkaisu

Eri tietoturvamalleihin tutustuttua päädyin eräänlaiseen pääsylistoihin perustuvaan sovellettuun malliin. Ratkaisumallissa resursseihin liitetään pääsylista, jolla määritellään kyseisen resurssin käyttöoikeudet käyttäjille. Pääsylista voi viitata käyttäjään suoraan tai epäsuorasti käyttäjäryhmän kautta.

4.2.1 Tietomalli

Kuviossa 1 on kuvattu suurpiirteisesti käyttöoikeusmallin komponenttien suhteet. Resurssityyppi määrittää resurssin kontekstin. Tällä tarkoitetaan sitä, että tiettyyn resurssityyppiin kuuluvat resurssit ovat käyttöoikeuksien ja käyttöympäristön suhteen samankaltaisia. Resurssityypille määritellään käyttöoikeudet, joita kyseisen resurssityypin resurssit käyttävät tai ymmärtävät. Tietyn resurssityypin resurssiin voidaan siis antaa vain niitä käyttöoikeuksia, joita kyseiseen resurssityyppiin on määritelty. Jokaiseen resurssiin liitetään pääsylista, jossa voi olla useampi pääsylista-alkio. Pääsylista-alkioon liitetään käyttöoikeus, joka voi olla ainoastaan pääsylistan kohteena olevan resurssin resurssityypin mukainen, sekä käyttäjä tai käyttäjäryhmä. Mikäli pääsylista-alkio viittaa käyttäjäryhmään, vaikuttaa se kaikkiin käyttäjäryhmään kuuluviin käyttäjiin.



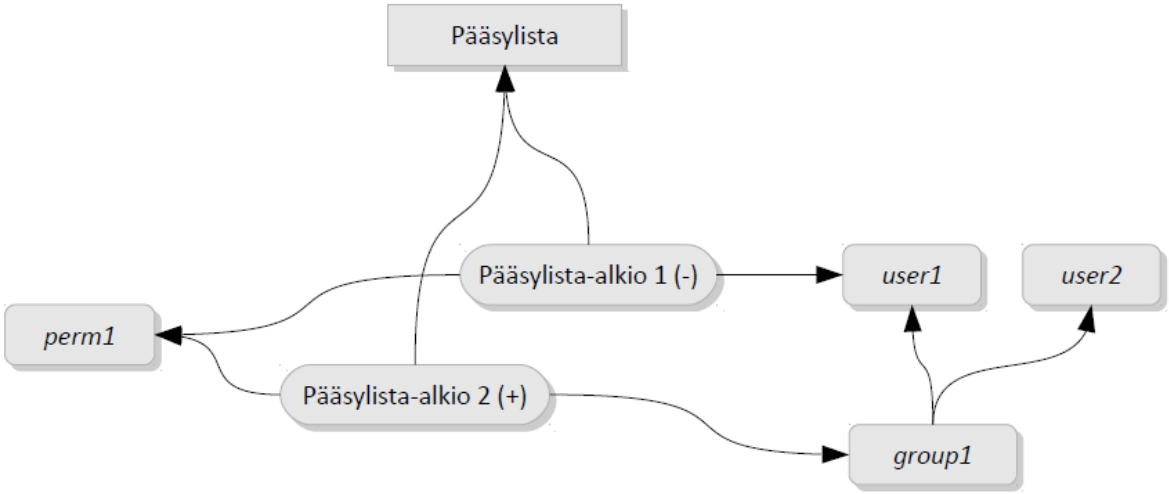
KUVIO 1. Käyttöoikeusmalli

Pääsylista voi siis koostua useammasta pääsylista-alkiosta. Alkiot muodostavat järjestetyn listan. Alkioiden järjestyksellä on merkitystä kun ratkaistaan käyttäjään vaikuttavat käyttöoikeudet, sillä pääsylistassa voi olla useampi alkio joka viittaa suoraan tai epäsuorasti samaa käyttäjään ja alkiot voivat olla joko sallivia tai epääviä. Suoralla viittauksella tarkoitetaan viittausta yksittäiseen käyttäjään, kun taas epäsuoralla viittausta ryhmään, jonka kautta saadaan lopulta assosiaatio käyttäjään. Salliva alkio antaa käyttäjälle alkiossa viitatun käyttöoikeuden kohteena olevaan resurssiin, kun taas epäävä alkio poistaa kyseisen käyttöoikeuden. Tällä tavoin voidaan luoda poikkeuksia esimerkiksi yksittäisille käyttäjille, kun käyttöoikeudet periytyvät kyseisille käyttäjille myös käyttäjäryhmien kautta.

4.2.2 Vaikuttavien käyttöoikeuksien selvitys

Kun käyttäjän käyttöoikeutta haluttuun resurssiin ratkaistaan, käydään pääsylista-alkioita läpi järjestyksessä, kunnes löydetään ensimmäinen käyttäjään viittaava alkio. Mikäli alkio on epäävä, ei käyttöoikeutta anneta käyttäjälle eikä listaa myöskään etsitä eteenpäin, jolloin listassa myöhempänä mahdollisesti käyttäjään viittaava saman käyttöoikeuden salliva alkio ei ole vaikuttava.

Tarkastellaan kuviossa 2 esitettyä esimerkkiä. Käyttäjät *user1* ja *user2* kuuluvat ryhmään *group1*. Resurssin pääsylistassa on kaksi alkiota, joista ensimmäinen evää käyttöoikeuden *perm1* käyttäjälle *user1* ja toinen sallii käyttöoikeuden *perm1* ryhmälle *group1*. Tällöin käyttäjällä *user1* ei ole käyttöoikeutta *perm1* kyseiseen resurssiin, koska ensimmäinen käyttäjään viittaava pääsylista-alkio on epäävä. Toisaalta käyttäjällä *user2* on käyttöoikeus *perm1*, koska ensimmäinen käyttäjään ryhmän *group1* kautta viittaavaa alkio on salliva.



KUVIO 2. Vaikuttavien käyttöoikeuksien selvitys pääsylistasta

Usein on tarpeen selvittää käyttäjän kaikki käyttöoikeudet tiettyyn resurssiin. Algoritmi käyttöoikeuksien selvittämiseksi etenee seuraavalla tavalla:

1. Koostetaan joukko S käyttäjästä u sekä jokaisesta käyttäjäryhmästä g, johon käyttäjä kuuluu.
2. Luodaan tyhjä käyttöoikeusjoukko P.
3. Käydään läpi käänteisessä järjestyksessä jokainen pääsylista-alkio a, joka on liitetty resurssin r pääsylistaan A sekä joukkoon S.
   3.1. Jos pääsylista-alkio a on salliva, lisätään sen viittaama käyttöoikeus p joukkoon P.
   3.2. Jos päästylista-alkio a on epäävä, poistetaan sen viittaama käyttöoikeus p joukosta P, mikäli se siihen kuului.

Joukossa P on tämän jälkeen kaikki resurssiin r vaikuttavat käyttöoikeudet käyttäjälle u. Koska pääsylista-alkiot käydään läpi käänteisessä järjestyksessä, on lopputulos se, että listan järjestyksessä ensimmäinen alkio on määräävin.

4.2.3 Resurssit ja resurssityypit

Resurssilla kuvataan asiaa, johon käyttöoikeuksia sovelletaan. Resurssityypillä määritellään resurssin konteksti eli se, minkätyyppinen resurssi on kyseessä. Koska samalla järjestelmällä hallitaan käyttöoikeuksia moniin hajautettuihin järjestelmiin, täytyy resursseihin pystyä viittaamaan yleispätevällä tavalla.

Päädyin ratkaisuun, jossa jokainen resurssi yksilöidään URI:lla (Uniform Resource Identifier[2]). Yleisesti ottaen URI on merkkijono, jolla kerrotaan jonkin tiedon sijainti. Esimerkiksi verkko-osoitteet (URL, Uniform Resource Locator[3]) ovat URI-tyyppisiä. Ratkaisun etuna on hierarkkisuus sekä se, että URI:t ovat tekstipohjaisia ja siten helpommin ymmärrettävissä ja ylläpidettävissä.

Hierarkkisuudesta saatava etu edellyttää käytäntöjä. URI:t täytyy siis muodostaa tietyllä sovitulla tavalla. Koska verkko-osoitteet ovat myös URI-tyyppisiä ja niiden käsittelyyn löytyy valmiit työkalut PHP:stä, päätin käytännöstä, että resurssitunnisteet ovat URL:ien kaltaisia.

Koska resurssityyppi on määräävin tekijä resurssissa, voidaan sen sanoa olevan hierarkiassa ylimpänä. On siten johdonmukaista käyttää resurssityyppiä tunniste-URL:n skeemana (engl. scheme). Esimerkiksi WordPress-resurssityypillä, jonka resurssit ovat WordPress-julkaisujärjestelmiä, voisi olla konekielinen resurssityypin nimi ”wordpress”. Tällöin kaikkien tätä resurssityyppiä olevien resurssien tunnisteet alkaisivat merkkijonolla ”wordpress:”.

Tunnisteen sisältöosa on skeemakohtainen. Esimerkiksi WordPress-resurssityypin sisältöosa voisi viitata kyseiseen WordPress-instanssiin. Sisältöosan ei tarvitse olla ns. vahva linkki kyseiseen WordPressiin, kunhan se yksilöi instanssin kyseisen käyttöoikeuksien hallinnan sisällä. Resurssin URI voisi siis olla kokonaisuudessaan vaikkapa ”wordpress://esimerkki-koulu”.

---

[2] Lisää aiheesta: http://tools.ietf.org/html/rfc3986.

[3] Lisää aiheesta: http://www.w3.org/Addressing/URL/url-spec.txt.

Kun WordPressiin myöhemmin tehdään lisäosa, joka tunnistaa käyttäjän OnEdun käyttäjienhallintajärjestelmää vastaan, voidaan lisäosan asetuksissa ilmaista, että kyseinen WordPress on nimeltään ”esimerkki-koulu”. Tämän tiedon avulla lisäosa voi kysyä käyttöoikeuksien hallinnasta, onko käyttäjällä *x* oikeus tehdä operaatio *y* resurssiin ”wordpress://esimerkki-koulu”.

5 TYÖSSÄ KÄYTETYT TEKNOLOGIAT

Opinnäytetyön ohjelmiston sovelluslogiikka toteutettiin PHP-kielellä. Tietokanta suunniteltiin käytettäväksi MySQL-ohjelmistossa.

Koska palvelu on verkkopohjainen, tarvitaan myös HTTP-palvelinohjelmisto. Kehityksen aikana käytettiin Apache HTTP Server -palvelinohjelmistoa. Se, mikä HTTP-palvelinohjelmisto lopulta valitaan tuotantoon, on triviaalia, koska PHP toimii itsenäisenä prosessina, jolle HTTP-palvelinohjelmisto välittää sivupyynnöt. Apache HTTP Server on kuitenkin osoittautunut vakaaksi ja nopeaksi vaihtoehdoksi.

Ohjelmistokehitys tapahtui Eclipse IDE -ohjelmistolla. Versiohallintaan käytettiin Git-versionhallintaohjelmistoa, jonka avulla hoidettiin myös lähdekoodien varmuuskopiointi. Varmuuskopiointi tapahtuu siten, että Git-säilö (engl. repository) sijaitsee erillisellä palvelimella, josta otetaan varmuuskopiot päivittäin. Kun muutokset tallennetaan versionhallintaan, päivittyvät ne myös kyseiselle palvelimelle.

OnEdun sisällönhallintajärjestelmänä toimii WordPress. WordPress on toteutettu PHP-kielellä, ja se käyttää MySQL-tietokantaohjelmistoa. Koska käyttäjähallinnan tulee integroitua WordPressin, täytyy Wordpressiin tehdä tätä tarkoitusta varten lisäosa.

5.1 PHP

PHP on skriptikieli, jota käytetään pääasiassa dynaamisten verkkosivujen luontiin. Kieli on saanut vaikutteista C-kielestä, Javasta sekä Perl:stä. (PHP: General Information.)

PHP mahdollistaa proseduraalisen sekä olio-pohjaisen ohjelmointiparadigman käytön. PHP 5.3 -version myötä tullut tuki anonyymeille funktioille, ns. lambdoille, mahdollistaa myös jossain määrin funktionaalisen ohjelmointiparadigman käytön. Lisäksi PHP 5.5 -versioon tuleva tuki generator-funktioille laajentaa funktionaalisen ohjelmoinnin mahdollisuuksia ennestään.

5.2 MySQL

MySQL on Oracle Corporationin omistama relaatiotietokantaohjelmisto. MySQL on saatavilla kaupallisena sekä avoimen lähdekoodin versiona GNU GPL -lisenssillä.

MySQL on suosituin avoimen lähdekoodin tietokantaohjelmisto (What is MySQL?). Se on myös hyvin tuettu useimmissa ympäristöissä ja ohjelmointikielissä. Muun muassa PHP:ssa on sisäänrakennettu tuki MySQL-tietokantayhteyksille.

5.3 Apache HTTP Server

Apache HTTP Server on ilmainen avoimen lähdekoodin HTTP-palvelinohjelmisto (About the Apache HTTP Server Project). Ohjelmiston tehtävä on käsitellä Internet-selaimelta tulevat sivupyynnöt ja palauttaa pyydetty verkkosivu käyttäjälle.

5.4 Eclipse

Eclipse on ilmainen avoimen lähdekoodin ohjelmointiympäristö, eli ns. IDE (integrated development environment). Ohjelmassa on omat työtilat eri ohjelmointikielille, kuten myös PHP:lle. Lisäksi Eclipseen saa Git-tuen lisäosan avulla, joten versionhallinta toimii suoraan ohjelman sisällä.

5.5 Git

Git on ilmainen avoimen lähdekoodin versionhallintajärjestelmä. Git on hajautettu, mikä tarkoittaa sitä, että useat henkilöt voivat itsenäisesti muokata versionhallinnassa olevia tiedostoja, ilman että muutokset täytyy aina tallentaa yhteen keskitettyyn säilöön (engl. repository). Tällöin ohjelmistokehittäjät voivat myös työskennellä yhteydettömässä tilassa. (Loeliger & McCullough 2012, 2.) Usein kuitenkin käytetään yhtä keskitettyä säilöä, johon tiedostoihin tehdyt muutokset liitetään (engl. merge).

Git kannustaa käyttämään kehityshaaroja (engl. branch). Haarojen luominen ja liittäminen on tehty lähtökohtaisesti nopeaksi operaatioksi ja niiden käyttö on pyritty saamaan mahdollisimman helpoksi.

5.6 WordPress

WordPress on ilmainen avoimen lähdekoodin julkaisujärjestelmä (About WordPress). WordPress kehitettiin alkujaan kevyeksi blogi-alustaksi, mutta suosion kasvun ja laajan kehittäjäyhteisön myötä järjestelmä on kehittynyt varteenotettavaksi verkkosivujen sisällönhallintajärjestelmäksi (CMS, engl. content management system).

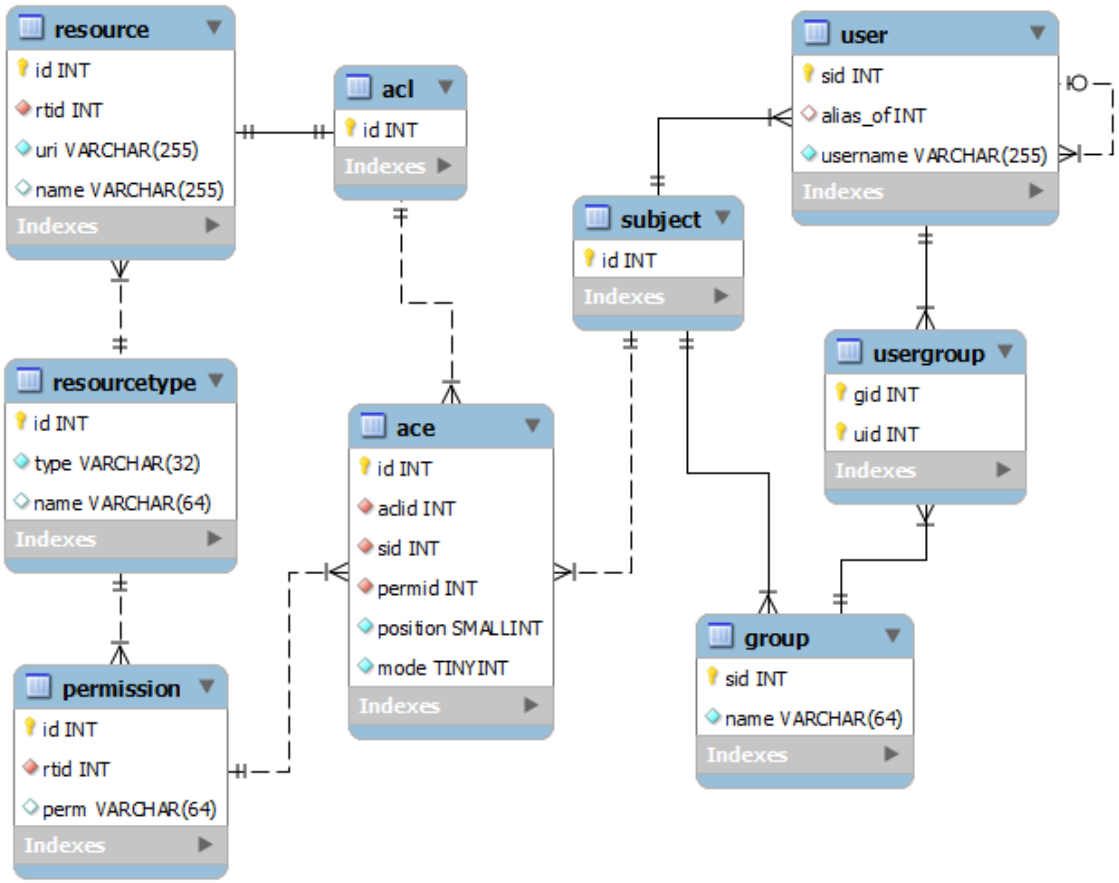
Wordpressin oleellinen ominaisuus on kattava tuki lisäosien, ns. pluginien kehittämiseen. Lisäosilla järjestelmää voi laajentaa ja tuoda siihen lisäominaisuuksia tarpeiden mukaan.

6 TIETOKANTA

Tässä luvussa esitellään käyttöoikeuksien tallennukseen suunniteltu tietokanta. Esitetty tietokantaskeema ei välttämättä ole lopullinen, eikä varsinkaan tauluissa ole määritelty kaikkia kenttiä. Tarkoitus on havainnollistaa kuinka valitun käyttöoikeusmallin mukaiset tiedot saadaan tallennettua pysyvään tietovarastoon.

Tietokanta on suunniteltu käytettäväksi MySQL-relaatiotietokantaohjelmistolla. Tietokantaskeeman suunnittelussa käytin MySQL Workbench -ohjelmistoa.

Kuviossa 3 on esitetty tietokantataulut ja niiden väliset relaatiot. Taulukossa 1 on selitetty lyhyesti taulujen merkitykset.



KUVIO 3. Tietokantaskeema käyttöoikeuksien tallentamiseen

TAULUKKO 1. Tietokantataulujen kuvaukset

Tietokantataulu	**Selitys**
resource	Resurssit, jotka ovat tiettyä resurssityyppiä.
resourcetype	Resurssityypit.
permission	Käyttöoikeudet, joita resurssityyppi voi saada.
acl	Resurssin pääsylista.

Tietokantataulu	Selitys
ace	Pääsylistan pääsylista-alkiot.
subject	Pääsylistan käyttäjäkohde.
user	Yksittäinen käyttäjä.
group	Käyttäjäryhmä.
usergroup	Käyttäjäryhmään kuuluvat käyttäjät.

Resurssi (*resource*-taulu) on abstrakti käsite, jolla viitataan mihin tahansa asiaan johon sovelletaan käyttöoikeuksia. Jokainen resurssi on tiettyä resurssityyppiä (*resourcetype*-taulu). Resurssityypeille määritellään käyttöoikeuksia (*permission*-taulu), joita kyseisen resurssityypin resursseille voidaan myöntää. Käyttöoikeudet ovat tekstimuotoisia tunnisteita, kuten vaikkapa *read* ja *write*.

Jokaiseen resurssiin liitetään pääsylista (*acl*-taulu). Vaikka tällä hetkellä pääsylistataulussa on ainoastaan *id*-kenttä, joka on myös vierasavain viitaten resurssitaulun *id*-kenttään ja täten ollen aina sama kuin resurssi-id, päädyin käyttämään pääsylistoille omaa taulua. Näin laajennettavuus tulevaisuudessa on helpompaa, mikäli pääsylistaan täytyy lisätä muita parametreja, tai jos halutaan että sama pääsylista voidaan liittää useampaan resurssiin.

Pääsylistaan liitetään 0..n kappaletta pääsylista-alkioita (*ace*-taulu, engl. access control entry). Pääsylista-alkiot on lajiteltu *position*-kentän mukaisesti. Lisäksi pääsylista-alkioon liitetään kohde (käyttäjä tai käyttäjäryhmä), käyttöoikeus sekä tieto siitä, onko alkio salliva vai epäävä (*mode*-kenttä).

Kohteella tarkoitetaan käyttäjää tai käyttäjäryhmää, jolle annetaan käyttöoikeus pääsylistan avulla. Jokaista käyttäjää (*user*-taulu) ja käyttäjäryhmää (*group*-taulu) varten luodaan merkintä erilliseen kohdetauluun (*subject*-taulu). Tämä on tarpeen jotta pääsylista-alkioista voidaan viitata sekä käyttäjiin että ryhmiin.

Käyttäjille voidaan luoda peitenimiä (engl. alias). Tämä voi tulla tarpeeseen, koska samoja käyttäjätunnuksia tullaan käyttämään useissa järjestelmissä, mutta käyttäjätunnus voi olla pakotettu tiettyyn muotoon. Esimerkiksi joku järjestelmä voi vaatia että käyttäjätunnus on sähköpostiosoite, mutta toinen järjestelmä ei kelpuuta erikoismerkkejä, kuten @. Peitenimellä voidaan siis luoda samalle käyttäjälle toinen käyttäjätunnus, mutta käyttöoikeuksia ei tarvitse ylläpitää kuin kyseisen käyttäjän ensisijaiselle tunnukselle. Käyttäjätaulun *alias_of*-kenttä on vierasavain saman tauluun. Mikäli kentän arvo ei ole tyhjä, käytetään tunnusta peitenimenä kyseiselle käyttäjälle.

Käyttäjä voi kuulua 0..n ryhmään. Käyttäjät liitetään ryhmiin ryhmäsidostaulun (*user-group*-taulu) avulla.

7 OHJELMISTOKEHYS

Tässä luvussa kuvataan käyttäjien ja käyttöoikeuksien hallintaa varten luotu ohjelmistokehys (engl. framework). Ohjelmistokehyksellä tarkoitetaan järjestelmän ydintä, jonka päälle rakennetaan varsinainen sovellus näkymineen ja business-logiikoineen.

Järjestelmän ohjelmistokehys on kokoelma luokkia (engl. class), jotka tarjoavat ydinpalvelut sekä rungon näkymille ja business-logiikalle. Järjestelmä on siis rakennettu olio-ohjelmoinnin periaatteiden mukaan. Jokainen luokka on kirjoitettu omaan tiedostoonsa ja tiedostot sijaitsevat tiedostojärjestelmän hakemistohierarkiassa. Luokat on jäsennetty nimiavaruuteen (engl. namespace) siten, että täydellisen luokan nimen (engl. fully qualified name, FQN) perusteella saadaan selville sen tiedoston polku, jossa luokka sijaitsee. Tätä käytäntöä hyväksikäyttäen on toteutettu luokkien automaattinen lataus.

Ohjelmistokehys tarjoaa MVC-arkkitehtuurin (malli-näkymä-käsittelijä, engl. model-view-controller) mukaisen ympäristön näkymien ja business-logiikan toteuttamiseen. Malleille on oma yhteinen perusluokka, josta kaikki mallit periytyvät. Mallien perusluokka tarjoaa operaatiot mallin kenttien alustukseen ja hallintaan. Samoin käsittelijöille on oma perusluokkansa. Käsittelijöiden perusluokka hoitaa perusoperaatiot näkymän alustamiseksi ja toimintojen sekä sivupyynnön käsittelemiseksi. Lisäksi ohjelmistokehys tarjoaa näkymäluokan, joka huolehtii sivupohjien tulostamisesta ja muuttujien välittämisestä.

7.1 Luokkakirjaston automaattinen lataus

PHP on dynaamisesti suoritettava skriptikieli. Tämä tarkoittaa sitä, että aina kun PHP-lähdekooditiedosto ajetaan, se parseroidaan ja käännetään PHP-tulkilla, minkä jälkeen se suoritetaan PHP-virtuaalikoneessa. Toisin kuin kielissä, joilla kirjoitetut ohjelmat käännetään kokonaisuudessaan ennen suoritusta, PHP-ohjelma ei suorituksen alussa tiedä, tarvitseeko se erillisiä tiedostoja ja missä ne sijaitsevat, vaan ohjelmoijan täytyy ottaa eri tiedostoissa sijaitsevat lähdekoodit mukaan `include`-funktiolla[4].

PHP tarjoaa ominaisuuden, jolla ohjelmassa voi rekisteröidä halutun funktion suoritettavaksi, kun yritetään käyttää luokkaa, jota ei ole vielä olemassa. Tätä ominaisuutta hyväk-

[4] `include` on itse asiassa PHP-kielen rakenneominaisuus eikä funktio sanan varsinaisessa merkityksessä. Lisää aiheesta: http://php.net/manual/en/function.include.php.

sikäyttäen voidaan toteuttaa luokkatiedostojen automaattisen latauksen logiikka, jolloin jokaista erillistä tiedostoa ei tarvitse sisällyttää lähdekoodiin `include`-funktiolla.

Automaattista latausta käyttämällä ratkaistaan kaksi ongelmaa, jotka vaikeuttavat laajempien PHP-kielisten ohjelmistojen kehitystä ja ylläpitoa. Ensinnäkin ohjelmoijan ei tarvitse huolehtia siitä, että kaikki tarvittavat tiedostot on sisällytetty lähdekoodiin, vaan luokat, ts. tiedostot, ladataan automaattisesti vasta tarpeen vaatiessa. Tekniikkaa kutsutaan usein laiskaksi lataukseksi (engl. lazy loading). Tämä vähentää myös virhealttiutta. Koska PHP-ohjelmaa ei käännetä kokonaisuudessaan ennen suoritusta, jolloin sen tarvitsemista puuttuvista tiedostoista saa virheilmoituksen vasta ajon aikana, saattaisivat kyseiset kriittiset ohjelmistovirheet ilmetä vasta tuotannossa. Toisekseen ohjelman vaatima muistintarve vähenee, koska kaikkia ohjelmiston tiedostoja ei tarvitse sisällyttää suoritukseen, vaikkei kyseisiä toimintoja edes tarvitsisi. Esimerkiksi MVC-mallin mukaiset käsittelijät, näkymät ja mallit ladataan vasta silloin, kun niitä toteuttaviin luokkiin viitataan lähdekoodissa ajon aikana ensimmäistä kertaa.

Automaattisen latauksen hoitava funktio rekisteröidään PHP-funktiolla `spl_autoload_register`[5]. Rekisteröintifunktiolle annetaan parametrina takaisinkutsufunktio (engl. callback), jota kutsutaan, kun ohjelman suorituksessa viitataan luokkaan, jota ei ole vielä olemassa.

Kuvassa 4 on esitetty automaattisen latauksen takaisinkutsufunktio. Funktio sijaitsee luokassa `Edukus`, joka on juurinimiavaruudessa, ts. luokan FQN on myös `Edukus`. `__CLASS__` on PHP:n ajonaikainen vakio (ns. magic constant[6]), joka pitää sisällään sen luokan nimen, jossa vakioon viitataan. Alussa tarkistetaan, että latausta vaativa luokka on samassa nimiavaruudessa, sillä muussa tapauksessa sitä ei osattaisikaan etsiä ja luokan lataus voidaan luovuttaa mahdollisesti toisen rekisteröidyn funktion suoritettavaksi. Seuraavaksi luokan FQN muutetaan tiedostojärjestelmän poluksi. Tämä tapahtuu korvaamalla kaikki FQN:n nimiavaruuserottimet (kenoviiva, `\`) tiedostojärjestelmän hakemistoerottimilla. Lisäksi alkuun lisätään hakemistopolku, jossa `Edukus`-luokka sijaitsee, sekä loppuun php-tiedostopääte. Näin esimerkiksi luokan `\Edukus\Util` tiedostopoluksi saadaan <perushakemisto>/Edukus/Util.php. Tämän jälkeen vielä tarkistetaan, että tiedosto on olemassa, luetaan se `include`-funktiolla ja vielä varmistetaan, että pyydetty luokka on tämän jälkeen saatavilla.

---

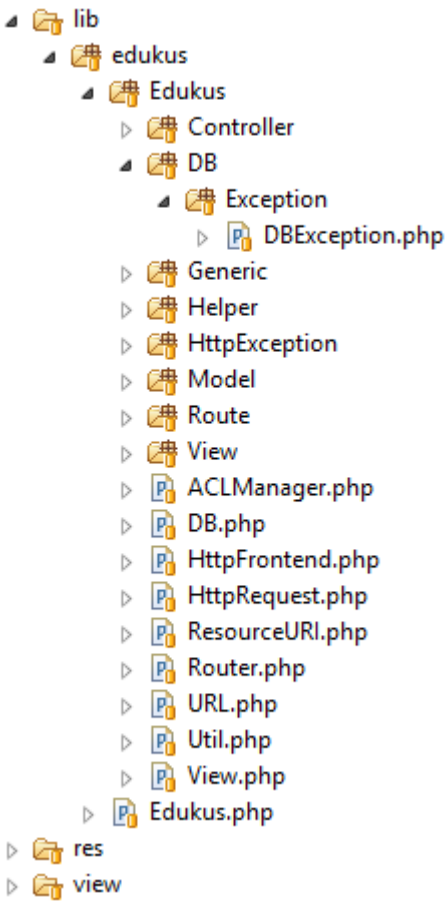
[5] Lisää aiheesta: http://php.net/manual/en/function.spl-autoload-register.php.

[6] Lisää aiheesta: http://php.net/manual/en/language.constants.predefined.php.

```php
/**
 * Include class of given name
 *
 * @param string $cls FQN of class to load
 * @return boolean
 */
public static function loadClass($cls) {
    static $prefix = __CLASS__;

    //check that requested class is in the same namespace
    if(substr($cls, 0, strlen($prefix)) !== $prefix) {
        return false;
    }

    $path = __DIR__ . DIRECTORY_SEPARATOR .
        str_replace("\\", DIRECTORY_SEPARATOR, $cls) . ".php";

    if(!is_file($path)) {
        return false;
    }

    if(!(include_once $path)) {
        return false;
    }

    if(!class_exists($cls, false)) {
        return false;
    }

    return true;
}
```

KUVA 4. Automaattisen latauksen takaisinkutsufunktio lataa lähdekooditiedoston, jossa pyydetty luokka on määritelty

Kuvassa 5 esitetystä hakemistohierarkiasta nähdään, kuinka luokkien nimiavaruudet ja tiedostojen polut toimivat yhteen. Esimerkiksi luokka `\Edukus\DB-\Exception\DBException` löytyy polusta <perushakemisto>/Edukus/DB/Exception-/DBException.php. Tätä käytäntöä noudattamalla automaattinen lataus löytää oikean tiedoston mille tahansa Edukus-nimiavaruuden luokalle.



KUVA 5. Lähdekoodien hakemistohierarkia

7.2 MVC-arkkitehtuuri

MVC eli malli-näkymä-käsittelijä (engl. model-view-controller) on ohjelmistokehityksessä laajalti käytetty arkkitehtuuri, jolla jäsennetään sovellus siten, että business-logiikka ja käyttöliittymät saadaan eriytettyä toisistaan (Pastor). MVC-arkkitehtuuri sopii hyvin verkkopohjaisiin sovelluksiin ja paradigmaa noudattamalla saadaan sovelluskehitys pidettyä järjestelmällisenä ja sovellus helpommin ylläpidettävänä.

Näkymissä ei tulisi suorittaa minkäänlaista tiedon prosessointia. Näkymien tehtävä on esittää niille välitetty valmis normaalimuotoinen data näkymän toteuttamassa muodossa. Esitysmuoto verkkopohjaisissa sovelluksissa on yleensä HTML.

Mallit huolehtivat datan käsittelystä. Koska verkkopohjaiset sovellukset ovat tilattomia, ts. tieto säilyy sovelluksen muistissa ainoastaan yhden sivupyynnön ajan, täytyy data tallentaa pysyvään tietovarastoon sivupyyntöjen väliseksi ajaksi. Usein käytetään erillistä tietokantaohjelmistoa, kuten MySQL:ää. Mallien tehtävä on ladata ja tallentaa data sovelluksen ja tietokannan välillä ja toimia rajapintana tietovarastoon.

Käsittelijät toimivat näkymien ja mallien välissä ohjaten sovelluksen toimintaa. Verkkopohjaisissa sovelluksissa käsittelijä lukee selaimelta sivupyynnössä saadut parametrit ja

toimii niiden mukaisesti. Käsittelijän tehtävänä on ladata tarvittavat mallit, valmistella esitettävä data normaalimuotoon, valita asianmukainen näkymä ja välittää esitettävä data näkymälle. Lisäksi käsittelijä hoitaa sovelluksen toimenpiteet, kuten uusien mallien luonnin ja mallien muokkauksen. Käsittelijä ei kuitenkaan ota kantaa siihen, missä muodossa tieto tallennetaan, vaan esimerkiksi uutta mallia luotaessa käsittelijässä ainoastaan syötetään käyttäjältä saadut arvot mallille ja pyydetään mallia tallentamaan itsensä pysyvästi.

7.2.1 Malli

Mallilla kuvataan sovelluksen dataa. Mallin tehtävä on huolehtia tiedon lataamisesta ja tallentamisesta pysyvään tietovarastoon sekä toimia rajapintana muulle sovellukselle.

Malleille tein perusluokan `AbstractModel`, joka tarjoaa perusoperaatioita mallin tietojen muokkaamiseen. Lisäksi perusluokka huolehtii mallin kenttien alustuksesta.

Perusluokasta periytyvät mallit määrittelevät omat kenttänsä normaaleina luokan ominaisuuksina (engl. properties). Kenttien nimet ovat käytännössä samat kuin mallia vastaavan tietokantataulun sarakkeiden nimet. Tämä yksinkertaistaa datan latausta ja tallennusta huomattavasti, koska tietokannalta saatavilla tiedoilla voidaan alustaa malli suoraan, samoin kuin mallilta saatavat tiedot voidaan suoraan syöttää tietokantaan, ilman että kenttien nimiä täytyy yksitellen eritellä.

Mallin kenttien ja tietokantataulun sarakkeiden yhtäläisyyttä noudattamalla voitiin perusluokkaan toteuttaa metodi `getFieldNames`, joka palauttaa listan mallin kentistä. Samat kentät tiedetään käytäntöä noudattamalla löytyvän myös mallia vastaavasta tietokantataulusta. `getFieldNames`-metodi käyttää kenttien nimien selvittämiseksi hyväksi PHP:n dynaamisen kielen ominaisuuksia.

Kuvassa 6 on esitetty `getFieldNames`-metodi. PHP:n `get_class`-funktio[7] palauttaa sille annetun luokan referenssin nimen. Koska luokan kentät eivät mallien tapauksessa muutu ajon aikana, tallennetaan kertaalleen selvitetty lista kentistä välimuistimuuttujaan. PHP:n `get_class_vars`-funktio[8] palauttaa assosiatiivisen taulukon annetun luokan kentistä ja niiden määreistä. Tästä tiedosta tarvitaan vain kenttien nimet, jotka saadaan ottamalla taulukosta vain arvojen avaimet funktiolla `array_keys`[9]. Lisäksi saadusta listasta

---

[7] Lisää aiheesta: http://php.net/manual/en/function.get-class.php.

[8] Lisää aiheesta: http://php.net/manual/en/function.get-class-vars.php.

[9] Lisää aiheesta: http://php.net/manual/en/function.array-keys.php.

suodatetaan pois kentät, jotka alkavat alaviivalla. Tämä käytäntö luotiin jotta mallit voivat määritellä omia sisäisiä muuttujia, jotka eivät kuitenkaan tallennu tietokantaan.

```
final public function getFieldNames() {
    static $cache = array();

    $cls = get_class($this);

    if(!isset($cache[$cls])) {
        //get all visible properties of given class
        $fields = array_keys(get_class_vars($cls));
        //filter properties that are prefixed with underscore
        $fields = array_filter($fields, function($f) {
            return substr($f, 0, 1) !== "_";
        });

        $cache[$cls] = $fields;
    }

    return $cache[$cls];
}
```

KUVA 6. `AbstractModel`-luokan `getFieldNames`-metodi palauttaa listan mallin kentistä

Listaa mallin kenttien nimistä käytetään useammassa luokan ominaisuudessa. Listan avulla mm. alustetaan mallin kentät tietokannasta saadulla rivi-datalla sekä koostetaan assosiatiivinen taulukko mallin arvoista, joka sitten voidaan suoraan tallentaa tietokantaan.

7.2.2 Näkymä

Näkymien tehtävä on esittää tieto loppukäyttäjälle. Näkymä itsessään ei suorita mitään tiedon prosessointia, vaan sille annetaan esitettävä data valmiiksi normalisoidussa muodossa. Normalisoinnilla tässä asiayhteydessä tarkoitetaan sitä, että tietojen oikeellisuutta ei tarvitse näkymätasolla tarkistaa eikä raakadatasta tarvitse suorittaa koostamista. Näkymässä voi tosin olla tarkoituksenmukaista iteroida sille annettua dataa, mikäli esitettävä tieto on taulukkomuotoista.

Verkkopohjaisissa sovelluksissa käytetään yleensä esityskerroksessa HTML-kieltä. Näkymä siis muodostaa annetusta datasta HTML-kielisen esityksen, joka sitten palautetaan HTTP-sivupyynnön vastauksena selaimeen. Lisäksi näkymä voi tulostaa HTML:n joukkoon JavaScript-koodia, CSS-määrittelyjä sekä muita selainpäässä suoritettavia tai käytettäviä ominaisuuksia.

Näkymien käsittelyyn tein `View`-luokan, jonka ilmentymä luodaan käsittelijässä. Käsittelijä voi asettaa `View`-luokan ilmentymään ominaisuuksia (engl. properties), ts. luokan jäsenmuuttujia. Koska PHP on dynaaminen kieli, ei luokkien ominaisuuksien tarvitse olla

valmiiksi määritelty, vaan niitä voidaan luoda ajon aikana. Data välitetään siis käsittelijältä näkymälle jäsenmuuttujien kautta.

Kun käsittelijä on alustanut näkymän, ts. prosessoinut datan ja asettanut sen pohjalta tarvittavat näkymän ominaisuudet, kutsutaan `View`-ilmentymän `render`-metodia. `render`-metodille välitetään sivupohjan nimi sekä valinnainen taulukko parametreja, jotka halutaan muuttujiksi sivupohjaan.

Sivupohjat itsessään ovat normaaleja PHP-tiedostoja, joissa on sekaisin HTML- sekä PHP-koodia. Sivupohjatiedostot suoritetaan `include`-funktiolla kaapaten niiden tuottama tuloste PHP:n output buffering -toiminnolla[10]. Suoritus tapahtuu `View`-luokan ilmentymän näkyvyysalueella (engl. scope), joten sivupohjissa voidaan käyttää `$this`-muuttujaa viittamaan näkymän jäsenmuuttujiin, jotka on asetettu käsittelijässä.

Kuvassa 7 on esitetty `render`-metodi sekä sen käyttämä `_render`-käärefunktio (engl. wrapper). Aluksi sivupohjan nimen perusteella selvitetään sivupohjatiedoston sijainti ja tarkistetaan että tiedosto on olemassa. Lisäksi tarkistetaan, onko valinnainen parametritaulukko annettu, ja mikäli ei, käytetään tyhjää taulukkoa. Ennen sivupohjan suoritusta käynnistetään tulosteen puskurointi `ob_start`-funktiolla, jolloin PHP:n tuottama tuloste ei päädy suoraan selaimeen, vaan se puskuroidaan muistiin. Itse sivupohjan suoritus tapahtuu kutsumalla `_render`-käärefunktiota, joka purkaa annetun parametritaulukon `extract`-funktiolla[11] `_render`-metodin näkyvyysalueelle paikallisiksi muuttujiksi sekä suorittaa sivupohjatiedoston `include`-funktiolla. Koska sivupohjan PHP-koodi suoritetaan `_render`-metodin näkyvyysalueella, voidaan sivupohjassa viitata parametritaulukon määrittelemiin muuttujiin. Lisäksi voidaan käyttää `$this`-muuttujaa, jonka avulla päästään käsiksi käsittelijässä alustettuihin näkymän jäsenmuuttujiin.

[10] Lisää aiheesta: http://php.net/manual/en/book.outcontrol.php.

[11] Lisää aiheesta: http://php.net/manual/en/function.extract.php.

```
public function render($tmpl, array $args = NULL) {
    $path = ViewFactory::getViewPath($tmpl);
    if(!is_file($path)) {
        return NULL;
    }

    $args = is_array($args) ? $args : array();

    ob_start();
    $this->_render($path, $args);

    return ob_get_clean();
}


protected function _render() {
    extract(func_get_arg(1));
    include func_get_arg(0);
}
```

KUVA 7. `View`-luokan `render`-metodi palauttaa näkymätulosteen annetusta sivupohjasta

7.2.3 Käsittelijä

Käsittelijän tehtävä on nimensä mukaisesti käsitellä toiminnot ja siten ohjata sovelluksen toimintaa. Käsittelijä toimii näkymien ja mallien välissä, eikä siten ota kantaa esitysmuotoon eikä tiedon tallennusmuotoon. Tyypillisessä verkkosovelluksen sivupyynnössä käsittelijä lataa tarvittavat mallit, tarvittaessa tarkistaa tietojen oikeellisuuden ja luo koosteita, välittää esitysvalmiin datan näkymälle ja käskee näkymää luomaan näkymänmukaisen esitysmuodon. Lisäksi käsittelijä voi suorittaa muita alustustoimenpiteitä, kuten käyttäjän valtuutuksen (engl. authorization) eli käyttöoikeuksien tarkistuksen.

Verkkosovelluksessa toiminnot määritellään useimmiten sivupyynnön osoitteella tai osoiteparametreilla. Toimintoja voisi olla esimerkiksi käyttäjän tietojen muokkauslomakkeen tulostus, käyttäjän tietojen tallennus ja käyttäjä poisto. Se, kuinka toiminnot jaotellaan käsittelijöiden kesken, riippuu pitkälti sovituista käytännöistä ja ohjelmistokehyksen toteutuksesta. Olen pitänyt nyrkkisääntönä että mikäli toiminnot käyttävät samaa näkymää, ne kuuluvat saman käsittelijän alle. Esimerkiksi käsittelijän, joka hoitaa käyttäjän tietojen muokkauslomakkeen tulostuksen, on loogista hoitaa myös toiminto, jolla päivitetään kyseiseltä lomakkeelta saadut tiedot. Tietojen tallennuksen jälkeen käsittelijä voi siten esittää saman lomakkeen päivitetyillä tiedoilla. Sen sijaan käyttäjän poisto on asianmukaista olla omassa käsittelijässään, koska poiston jälkeen ei voida enää esittää lomakenäkymää, sillä käyttäjää ei ole olemassa.

Käsittelijöille tein perusluokan `AbstractController`, josta kaikki käsittelijät periytyvät. Lisäksi lopullisten käsittelijöiden ja perusluokan välisessä luokkahierarkiassa on `Base-`

AdminController-luokka, josta hallintapuolen käsittelijät periytyvät. BaseAdminController hoitaa alustukset ja käyttäjän valtuutukset, jotka ovat yhteisiä kaikille hallintapuolen toiminnoille.

Perusluokalla on muutamia metodeja, jotka on tarkoitus korvata (engl. override) lopullisissa käsittelijäluokissa. isAuthorized-metodin tulee palauttaa totuusarvo tosi, mikäli istunnon käyttäjällä on käyttöoikeudet pyydetyn käsittelijän käyttöön. Mikäli käyttäjällä on todettu olevan valtuudet kyseisen käsittelijän käyttöön, kutsutaan init-metodia, jossa käsittelijä voi alustaa tarvittavat mallit ja asettaa tarvittavat jäsenmuuttujat näkymälle. Perusluokan getView-metodi instantioi kutsuttaessa näkymäluokan käyttäen ns. laiskaa latausta ja palauttaa View-luokan ilmentymän. Myös init-metodin täytyy palauttaa totuusarvo tosi merkiksi siitä, että käsittelijän alustus on suoritettu onnistuneesti.

Perusluokan processRequest-metodi huolehtii käsittelijän suorituksesta tarkistaen valtuutuksen ja kutsuen lopullisen käsittelijäluokan alustusfunktiota. Mikäli jokin metodi palauttaa totuusarvon epätosi, nostetaan sen mukainen poikkeus (engl. exception). Esimerkiksi jos käyttäjän käyttöoikeudet eivät riitä, nostetaan UnauthorizedControllerException-poikkeus, joka käsitellään matalammalla tasolla ohjelman suoritusta näyttäen virhesivu.

Mikäli käsittelijän alustus suoritetaan onnistuneesti, siirrytään toiminnon suorittamiseen. Toiminnot välitetään käsittelijälle HTTP POST -metodilla, *action*-parametrissa. Toimintoja vastaamaan täytyy luoda metodi käsittelijään, jonka nimi on muotoa action_<toiminto>. Esimerkiksi tallennuksesta vastaava toiminto nimeltä ”save” tarvitsee action_save-metodin toimiakseen.

Mikäli toimintoa ei ole erikseen sivupyynnössä annettu, ei kutsuta mitään toimintometodia. Toisaalta voidaan sanoa, että tällöin suoritetaan vakiotoiminto ”view”, joka ainoastaan esittää käsittelijän mukaisen näkymän. Esimerkiksi käyttäjätietojen muokkausnäkymä UserEditController esittää aina pyydetyn käyttäjän tietojenmuokkauslomakkeen, vaikkei erityistä toimintoa olisikaan määritelty. Kun lomake selaimesta lähetetään, siirtyy mukana myös action-parametrissa arvo ”save”, joka aiheuttaa käsittelijän tallennuslogiikan suorituksen.

Kuvassa 8 on esitetty _processActions-metodi, joka huolehtii toimintoja vastaavien metodien kutsumisesta. Funktio käyttää HttpRequest-luokan singleton-ilmentymää sivupyynnön parametrien lukemiseen. Tein HttpRequest-luokan yksinkertaistamaan ja yhtenäistämään toimintoja, jotka liittyvät HTTP-protokollan sivupyynnön käsittelyyn. Aluksi tarkistetaan että sivupyyntö on tehty HTTP POST-metodilla. Sitten tarkistetaan että *action*-parametri on välitetty selaimelta sovellukselle. _actionToMethod-funktio muut-

taa toiminnon nimen käsittelijän metodin nimeksi huolehtien myös validoinnista, ts. ettei käyttäjän ole mahdollista kutsua mielivaltaisia metodeja näin aiheuttaen tietoturvariskin. Mikäli toiminnon mukainen metodi on määritelty, kutsutaan sitä ja palautetaan totuusarvo tosi.

```
private function _processActions() {
    $request = HttpRequest::getInstance();

    if($request->getMethod() !== "POST") {
        return false;
    }

    $action = $request->getPostParam("action");
    if($action === NULL) {
        return false;
    }

    $mtd = self::_actionToMethod($action);
    if(!is_callable(array($this, $mtd))) {
        return false;
    }

    $this->$mtd();

    return true;
}
```

KUVA 8. `AbstractController`-luokan `_processActions`-metodi huolehtii käsittelijän suorituksesta

7.3 Reititys

Reitityksellä tarkoitetaan prosessia, joka selvittää minkä käsittelijän tehtävä on hoitaa mikäkin sivupyyntö. Reititysprosessista vastaa `Router`-luokka, joka käy läpi esimääritellyn listan reittejä. Reitit ovat ilmentymiä luokista, jotka periytyvät `AbstractRoute`-perusluokasta. `AbstractRoute`-luokasta periytyvät luokat toteuttavat `match`-metodin, joka saa parametrina sivupyynnön verkko-osoitteen. Mikäli verkko-osoite täsmää kyseiseen reittiin, palauttaa `match`-metodi totuusarvon tosi. Reititysprosessi lopetetaan kun ensimmäinen täsmäävä reitti löytyy. Tämän jälkeen pyydetään täsmänneeltä reitiltä sen mukainen käsittelijäluokan ilmentymä, joka jatkaa sivupyynnön käsittelyä.

Reittityyppejä tein kolmenlaisia. Kaikki reittityypit periytyvät `AbstractRoute`-perusluokasta.

`StaticRoute`-luokka täsmää yksinkertaiseen staattiseen verkko-osoitteen polkuun, kuten /index. Haluttu polku annetaan luokan muodostinfunktiolle (engl. constructor).

`RegexRoute`-luokalla voi täsmätä verkko-osoitteen polkuun käyttämällä säännönmukaista lauseketta (engl. regular expression). Säännönmukaisen lausekkeen tallennetut ryhmät (engl. capturing groups) voidaan lisäksi välittää käsittelijälle sivupyynnön parametreina.

`DynamicRoute`-luokkaa käytetään reittilistan lopussa ns. *fallback*-reittinä, joka käsittelee sivupyynnöt joille ei ole määritetty erityistä reittiä. Luokka muodostaa dynaamisesti käsittelijän nimen verkko-osoitteen kahdesta ensimmäisestä polkukomponentista. Esimerkiksi /user/edit/13 polusta saadaan käsittelijäluokan nimi `UserEditController`. Mikäli kyseinen luokka on olemassa, reitti täsmää ja `match`-metodi palauttaa arvon tosi.

Reitit määritellään `HttpFrontend`-luokan `handleRequest`-metodissa, joka on ensimmäinen kutsuttava funktio automaattisen latauksen rekisteröimisen jälkeen. Metodissa rekisteröidään reitit `Router`-luokan singleton-ilmentymälle, jonka jälkeen pyydetään `Router`-luokkaa suorittamaan reititys. Mikäli yksikään reitti ei täsmää sivupyyntöön, näytetään ”HTTP 404 Not Found” -virhesivu.

Kuvassa 9 on esitetty `route`-metodi, joka etsii ensimmäisen täsmäävän reitin. Ensiksi pyydetään sivupyynnön osoite `HttpRequest`-luokan `getUrl`-metodilta. Tämän jälkeen käydään läpi kaikki rekisteröidyt reitit järjestyksessä ja kutsutaan reittiluokkien `match`-metodia, kunnes ensimmäinen täsmää. Mikäli yksikään reitti ei täsmää, palautetaan tyhjä arvo NULL.

```
public function route() {
    $requri = HttpRequest::getInstance()->getURL();
    foreach($this->_routes as $route) {
        if($route->match($requri)) {
            return $route;
        }
    }

    return NULL;
}
```

KUVA 9. `Router`-luokan `route`-metodi etsii rekisteröidyistä reiteistä ensimmäisen joka täsmää sivupyyntöön

8 YHTEENVETO

Opinnäytetyö oli rajattu OnEdun käyttäjien ja käyttöoikeuksien hallinnan suunnitteluun sekä ohjelmistokehyksen toteutukseen. Suunnitteluun kuului keskeisesti ongelmaan soveltuvan käyttöoikeusmallin kehittäminen. Lisäksi suunnitteluvaiheessa täytyi kartoittaa mahdollisimman kattavasti tulevaisuuden tarpeet, jotta niihin voidaan tulevassa ohjelmistokehityksessä varautua.

Rajaus tehtiin, koska jo projektin alussa kokonaisuuden tiedettiin työmäärältään olevan täysin epärealistinen opinnäytetyölle varatulle ajalle. Lisäksi opinnäytetyötä tehtiin täyspäiväisen ansiotyön ohessa, mikä asetti omat haasteensa ajankäytölle. Opinnäytetyö saatiin kuitenkin tehtyä sovitussa laajuudessaan määräaikaan mennessä.

Opinnäytetyölle asetetut tavoitteet saavutettiin mielestäni hyvin. Kuvattuun ongelmaan syventyminen auttoi vertailemaan eri ratkaisumalleja ja löytämään järkevimmältä tuntuvan kompromissin. Lopullinen toteutus ja käytäntö tulevat näyttämään, oliko valinta oikea.

Ohjelmistokehyksen luomisessa käytin ammattiosaamistani sekä osin vanhoista projekteista otettuja yleiskäyttöisiä ohjelmistoluokkia. Tältä osin työ ei ollut kovinkaan haastava eikä aiheuttanut ongelmia, vaan eteni rutiinilla hyvin nopeasti.

Ohjelmiston kehitys jatkuu ketterän ohjelmistokehityksen periaatteita noudattaen. Ensimmäisessä vaiheessa on tärkeintä saada määriteltyä rajapinnat, jotta integraatiomoduuleja voidaan alkaa rakentaa ulkoisiin järjestelmiin.

LÄHTEET

About the Apache HTTP Server Project [verkkosivu]. The Apache Software Foundation [viitattu 27.4.2012]. Saatavissa: http://httpd.apache.org/ABOUT_APACHE.html.

About WordPress [verkkosivu]. [viitattu 27.4.2013]. Saatavissa: http://wordpress.org/about/.

An Introduction to Role-Rased Access Control [verkkosivu]. National Institute of Standards and Technology [viitattu 27.4.2013]. Saatavissa: http://csrc.nist.gov/groups/SNS/rbac/documents/design_implementation/Intro_role_based_access.htm.

Loeliger, J. & McCullough, M. 2012. *Version Control with Git*. 2. painos. Sebastopol: O'Reilly Media.

Pastor, P. MVC for Noobs [verkkosivu]. 24.3.2010 [viitattu 14.3.2013]. Saatavissa: http://net.tutsplus.com/tutorials/other/mvc-for-noobs/.

PHP: General Information [verkkosivu]. The PHP Group [viitattu 12.3.2013]. Saatavissa: http://www.php.net/manual/en/faq.general.php.

What is ACL? [verkkosivu]. Webopedia.com [viitattu 27.4.2013]. Saatavissa: http://www.webopedia.com/TERM/A/ACL.html.

What is MySQL? [verkkosivu]. Oracle [viitattu 12.3.2013]. Saatavissa: http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html.